(19) Canadian Intellectual Property Office
An Agency of Industry Canada

Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

(11) CA 2 255 054 (13) A1
(43) 30.05.2000

(12)

(21) 2 255 054

(51) Int. Cl.⁶: G06F 009/44, G06F 009/45

(22) 30.11.1998

(71) IBM CANADA LIMITED-IBM CANADA LIMITEE, 3600 Steeles Avenue East, MARKHAM, O1 (CA).

(72) SLUIMAN, Harm (CA). LAU, Christina P. (CA).

(74) HOFFMAN, JEAN-PAUL

(54) METHODE ET MECANISME RELATIFS A UN MODELE DE DONNEES ORIENTE TACHES
(54) METHOD AND MECHANISM FOR A TASK ORIENTED DATA MODEL

(57)

A task oriented data model for an object oriented development tool. The data model comprises a task oriented structure which mirrors the task tree embodied in the user interface of the development tool. The object development tool exports and imports the data model as a document file expressed in a open standard language such as XML. The data model comprises a series of data elements which are arranged to mirror the user interface according to a meta data model. The meta data model is implemented in XML as a Document Type Definition and comprises containment relationships defined by XML constructs. By making the user interface implicit in the organization of the data model, the data model can be imported by another tool by simply following or scripting the data structure, and the data model is validated when the import utility loads the document file by comparing the data model to a meta data model specified in the Document Type Definition.

OPIC
OFFICE DE LA PROPRIÉTÉ INTELLECTUELLE DU CANADA

CIPO
CANADIAN INTELLECTUAL PROPERTY OFFICE

(12) (19) (CA) **Demande-Application**

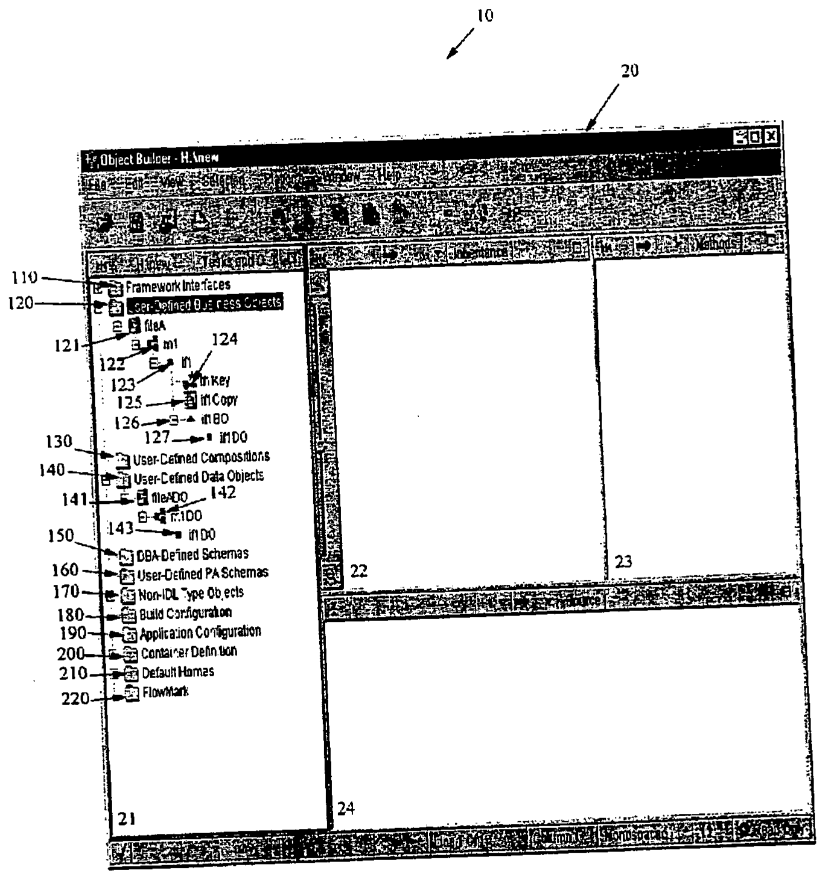
(21) (A1) **2,255,054**
(22) 1998/11/30
(43) 2000/05/30

(72) LAU, Christina P., CA
(72) SLUIMAN, Harm, CA
(71) IBM CANADA LTD. - IBM CANADA LIMITÉE, CA

(51) Int.Cl.^{6} G06F 9/44, G06F 9/45

(54) **METHODE ET MECANISME RELATIFS A UN MODELE DE DONNEES ORIENTE TACHES**

(54) **METHOD AND MECHANISM FOR A TASK ORIENTED DATA MODEL**



(21) (A1) 2,255,054
(22) 1998/11/30
(43) 2000/05/30

(57) A task oriented data model for an object oriented development tool. The data model comprises a task oriented structure which mirrors the task tree embodied in the user interface of the development tool. The object development tool exports and imports the data model as a document file expressed in a open standard language such as XML. The data model comprises a series of data elements which are arranged to mirror the user interface according to a meta data model. The meta data model is implemented in XML as a Document Type Definition and comprises containment relationships defined by XML constructs. By making the user interface implicit in the organization of the data model, the data model can be imported by another tool by simply following or scripting the data structure, and the data model is validated when the import utility loads the document file by comparing the data model to a meta data model specified in the Document Type Definition.

METHOD AND MECHANISM FOR A TASK ORIENTED DATA MODEL

ABSTRACT OF THE DISCLOSURE

A task oriented data model for an object oriented development tool. The data model comprises a task oriented structure which mirrors the task tree embodied in the user interface of the development tool. The object development tool exports and imports the data model as a document file expressed in a open standard language such as XML. The data model comprises a series of data elements which are arranged to mirror the user interface according to a meta data model. The meta data model is implemented in XML as a Document Type Definition and comprises containment relationships defined by XML constructs. By making the user interface implicit in the organization of the data model, the data model can be imported by another tool by simply following or scripting the data structure, and the data model is validated when the import utility loads the document file by comparing the data model to a meta data model specified in the Document Type Definition.

METHOD AND MECHANISM FOR A TASK ORIENTED DATA MODEL

REFERENCE TO RELATED APPLICATIONS

This application is related to the co-pending application entitled Method and Mechanism for a Task Oriented XML Data Model filed in the name of the common assignee.

FIELD OF THE INVENTION

The present invention relates to object oriented programming systems, and more particularly to a task oriented structure for a data model and method for validating the data model.

BACKGROUND OF THE INVENTION

Traditionally in the computer programming art, program code and data have been kept separate. For example, in the well-known C programming language, units of program code are called functions, and units of data are called structures. In C the functions and structures are not formally connected, that is, a function can operate on more than one type of data structure, and more than one function can operate on the same structure.

Object-oriented programming, in contrast, deals with objects, in which program code and data are merged into a single indivisible entity or "object". An object is an instance of an implementation and an interface and models a real-world entity. The object is implemented as a computational entity that encapsulates state and operations and responds to requests. The data and methods are implemented internally in the object. A business object for example is an object containing business methods (i.e. logic) and state that is intended for use within business applications.

Objects are built using an application program commonly referred to as an object builder tool. The object builder tool is a program which is used to define the business objects and generate the necessary definition and implementation files in IDL (Interface Definition Language), C++ and/or Java, as will be understood by those skilled in the programming arts.

Object builders utilize a data model which provides a template of the data structure for the objects. Object-oriented data models can be adapted to a wide range of different models by changing the definitions of meta-data, such as the objects and the task trees. Meta-data, as will be understood by those skilled in the art, comprises the self-descriptive information that can describe both services and information. With meta-data new services can be added to a system and discovered at run-time.

The object builder in the Component Broker™ toolkit from IBM utilizes a proprietary data model known as the Common Data Model (CDM) which comprises a binary file. In the object builder tool, objects are designed by performing sets of user-defined operations in the form of task trees.

Frequently, users will want to load an application program with data by means other than the user interface for the application. For example, in a spreadsheet program, a user may wish to load a subset of data extracted from a database. To be able to perform such a data load or import, an alternate user interface or an import capability will typically be required in order to translate the extracted subset of data into a form suitable for the spreadsheet program. The alternate user interface or import capability is needed to check the data for constraints that must be applied in order not to break the application. Typically, this involves an expensive piece of development work done by the provider of the application program. As a result, such a load capability is not usually provided.

BRIEF SUMMARY OF THE INVENTION

What is provided according to the present invention is a data structure which allows the data import facility in an application program to be simplified.

According to one aspect of the invention, the data structure comprises a task oriented data structure. The task oriented data structure is specified according to a hierarchical data model for the application. By providing the data in a task oriented structure, the data provides scripting for the import facility and allows the data to be validated according to the hierarchical data model when it is loaded into the application.

According to another aspect of the invention, the data model is expressed in an exportable and importable document in meta-data language such as XML, i.e. Extensible Markup Language. The data model expressed as an XML document is arranged in a task oriented structure which mirrors the user interface of the application (e.g. an object builder). The data model is specified according to a meta data model which takes the form of one or more Document Type Definitions (DTD's) in XML. The meta data model, i.e. DTD, specifies the set of required and optional data elements, and their attributes, for the data model expressed as an XML data document. According to the invention, the data model specified by the DTD is arranged in a hierarchical structure which represents the task flow or sequence that is followed by a user to enter the data.

To export a data model, the data model is expressed as a structured XML document file which mirrors the exact order in which the object data model was created by the user. Because the tasks are expressed in terms of containment relationships in the XML document as defined by the DTD, the importation, i.e. reading, of the XML document serves to extract the data model in the same sequence the data was entered. It will be appreciated that this approach eliminates the need for tags and other indicators of location as is the case for conventional data export/import files.

To import a data model, i.e. an XML document file, the export/import utility includes an XML parser. The XML parser parses the XML document and builds a "document tree" which is then returned to the data import/export utility. Because the XML document file, i.e. the data model, is arranged as a task oriented structure, the document file provides scripting for translating each data element contained in the data model. In addition, the XML document file, i.e. the data model, is validated when the import utility loads the document file by comparing the data model to a meta data model specified in the form of the DTD.

In accordance with an aspect of the invention, there is provided a task oriented data structure embodied in a program storage device for an object oriented application program, the object oriented application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating one or more objects, said data structure comprising: (a) a data model suitable for storage on a storage media; (b) said data model including a plurality of data elements; (c) each of said data elements corresponding to one of the tasks in said sequence of tasks; and (d) said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface. Said meta data model may include means for validating each of said data elements and the arrangement of said data elements. Said data model may also be expressed in an Extensible Markup Language or XML, and said data elements being arranged according to containment constructs specified in said meta data model. And, said means for validating may comprise a Document Type Definition specified in XML.

There is also provided, in an application program for creating objects, the application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating the objects, and said application program including an import utility for importing document files, a import utility comprising: (a) means for inputting a document file expressed in a meta data programming language, and wherein said document file comprises a plurality of data elements, each of said data elements corresponding to one of the tasks in said sequence of tasks, and said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface; (b) parser means for parsing said document file according to said meta data programming language, and said parser means including means for creating a document tree, said document tree comprising a plurality of nodes, wherein each of said nodes corresponds to an item in the user interface; and (c) means for updating the object according to the nodes contained in said document tree. Said meta data model may also include means for validating each of the nodes in said document tree. Said document file may further comprise a text file expressed in Extensible Markup Language or XML, and said data elements being arranged according to containment constructs specified in said meta data model. Said means for validating may also comprise a Document Type Definition specified in XML. And, said document file may include scripting means for translating each of said data elements.

In accordance with another aspect of the invention, there is provided a computer program product for an application program for creating objects, the application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating the objects, and said application program including an import utility for importing document files, said computer program product comprising a program storage device; means recorded on said program storage device for instructing a computer to perform the steps of, (a) inputting a document file, wherein said document file is expressed according to a meta data programming language, said document file comprising a plurality of data elements, each of said data elements corresponding to one of the tasks in said sequence of tasks, and said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface; (b) parsing said document file according to said meta data programming language; (c) creating a document tree from said parsed document file, said document tree comprising a plurality of nodes, wherein each of said nodes corresponds to an item in the user interface; and (d) updating the object according to the nodes contained in said document tree. Said computer program product further including validating means for use by said parsing means for validating each of the nodes in said document tree. Said meta data language may comprise Extensible Markup Language or XML, and said data elements being arranged according to containment constructs specified in said meta data model. And, said means for validating may comprise a Document Type Definition specified in XML.

There is also provided a computer system for creating objects in an application program, the application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating the objects, and said application program including an import utility for importing document files, said computer system comprising: (a) means for inputting a document file, wherein said document file is expressed according to a meta data programming language, said document file comprising a plurality of data elements, each of said data elements corresponding to one of the tasks in said sequence of tasks, and said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface; (b) a parser for parsing said document file according to said meta data programming language; (c) means for creating a document tree from said parsed document file, said document tree comprising a plurality of nodes, wherein each of said nodes corresponds to an item in the user interface; and (d) means for updating the object according to the nodes contained in said document tree. The computer system may also include validating means for use by said parser for validating each of the nodes in said document tree. Said meta data language may comprise Extensible claim Markup Language or XML, and said data elements being arranged according to containment constructs specified in said meta data model. Said validating means may comprise a Document Type Definition specified in XML. And, said data model may include scripting means for translating each of the data elements in said document file.

BRIEF DESCRIPTION OF THE DRAWINGS

Reference will now be made to the accompanying drawings which show, by way of example, a preferred embodiment of the present invention, and in which:

Fig. 1 is a diagrammatic representation of the main window of an object builder of the type suitable for a task oriented data model according to the present invention; and

Fig. 2 is a flow chart showing the processing steps performed by a data import utility to load a task oriented data model according to the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention comprises a task oriented data structure for use with a data import and export utility in an application program.

According to one embodiment of the invention, the task oriented data model is implemented using the open standard language XML (Extensible Markup Language). The data model is expressed as a structured XML document which is specified according to a meta data model contained in a Document Type Definition or DTD. The meta data model specified in the Document Type Definition is also arranged as a hierarchical structure which mirrors the user interface task flow sequence in the application program. The XML data document is compatible with the export and import utility in development tools (e.g. object builder tool). In the following description, the task oriented structure is described in the context of an object builder tool which is in the "toolkit" for the IBM Component Broker™ software package.

Reference is first made to Fig. 1, which shows in diagrammatic form the object builder tool 10 in the Component Broker toolkit. The object builder tool 10 is an application program which provides an integrated development environment for software developers. As shown in Fig. 1, the object builder 10 comprises a main window 20 having four panes denoted by 21, 22, 23, and 24 respectively. The first pane 21 on the left side of the window 20 provides the main view which is a task tree view. The main view in the first pane 21 is also the context for the content displayed in the three other panes 22, 23, and 24.

As shown in Fig. 1, the main view pane 21 comprises a "Framework Interfaces" item 110, a "User-Defined Business Objects" item 120, a "User-Defined Compositions" item 130, a "User-Defined Data Objects" item 140, a "DBA-Defined Schemas" item 150, a "User-Defined PA Schemas" item 160, a "Non-IDL Type Objects" item 170, a "Build Configuration" item 180, an "Application Configuration" item 190, a "Container Definition" item 200, a "Default Homes" item 210, and a "FlowMark" item 220. The content of the other three panes 22, 23 and 24 in the window 10 depends on the selected item or element in the main view pane 21.

It will be appreciated that the object builder 10 utilizes a task tree structure. As shown in Fig. 1, the "User-Defined Business Objects" item 120 has a user interface which is organized as a task tree structure. The task oriented structure of the user interface forces a user to follow a specified sequence in order to create a business object, i.e. an object containing business methods (logic) and data (state) that is intended for use within business applications. As will now be described, the data model and the underlying meta data model according to the present invention comprise task oriented structures which mirror the user interface. The data model includes all the persistent meta-data which is used by the object builder tool 10.

Referring to Fig. 1, and specifically the main view pane 21, the task tree structure for the "User-Defined Business Objects" item 120 comprises a file item or element 121, a module element 122, an interface element 123, a key helper element 124, a copy helper element 125, a business object implementation element 126, and a data object element 127. The "User-Defined Data Objects" item 140 is also organized in a task tree structure and comprises a file element 141, a module element 142, and a interface element 142. Similar task tree structures are provided for the "DBA-Defined Schemas" item 150, the "Non-IDL Types Objects" item 170, the "Build Configuration" item 180, the "Application Configuration" item 190, and the "Container Definition" item 200.

According to the task tree structure for the "User-Defined Business Objects" item 120, a business object is created by the following sequence of steps: (1) create file; (2) create module; (3) create interface; (4) create key helper; (5) create copy helper; and (6) create implementation.

The first step performed by a user involves creating a file. The object builder 10 generates the file as an IDL (Interface Definition File). As will be understood by one skilled in the art, IDL is a contractual, neutral and declarative language that specifies an object's boundaries and its interfaces, e.g. the aspects of the object that are accessible to the public. IDL provides operating system and programming language independent interfaces to all services and components that reside on a bus as defined according to the CORBA (Common Object Request Broker Architecture) standard. The second step is optional and allows a user to define a module which comprises a grouping of interfaces. The third step performed by the user involves defining the public interface for the user-defined business object. The fourth step involves creating a key helper which comprises an object which provides a copy of the "identity" of the business object. The fifth step involves creating an object which duplicates everything about the business object, not just its identity. The sixth step comprises implementing the specific behaviour, i.e. methods, for the business object.

According to the present invention, the data model comprises a task oriented structure which preserves the sequence of steps followed by a user during the creation of the object. The data model for an object, i.e. instance of data, is expressed by the object builder tool in the form of an XML document which mirrors the exact order in which the object was created by the user. The underlying meta data model also mirrors the task oriented or hierarchical structure of the user interface. The data model thereby provides a data structure from the which the user interface of the object builder tool can be inferred. Since the user interface is implicit in the organization of the data model, the data model can be imported by another tool by simply following or scripting the data model.

According to another aspect of the invention, the task oriented data model is expressed as a document file using the meta-data language XML (i.e. Extensible Markup Language) and the meta data model takes the form of Document Type Definition or DTD also specified in XML. A meta-data language is self-descriptive of both services and information and is preferred because the data model can be generically validated by reading or importing the data model. The meta data model, i.e. Document Type Definition or DTD, is used for validating the correctness of the data model which is expressed as an XML document.

According to this aspect of the present invention, the meta data model for the "User-Defined Business Objects" item 120 takes the form of a Document Type Definition which is specified as shown below.

```
<?xml encoding="US-ASCII"?>

<!ELEMENT UserDefinedBusinessObject (BusinessObjectUnit)*>
<!ELEMENT UserDefinedDataObject     (DOFile)*>
<!ELEMENT BusinessObjectUnit        (BOFile | CompositeFile)*>
<!ELEMENT DBADefinedSchemas         (SchemaGroup)*>
<!ELEMENT UserDefinedPAOs           (Schema)*>

<!-- ************************** BO File *************************** -->
<!ELEMENT BOFile (Comments| IncludeFile| Module | BOArtifact |
Constant | Enumeration | Exception | Structure | Typedef | Union | Uuid)*>
<!ATTLIST BOFile Name CDATA #IMPLIED>
<!ELEMENT Comments (#PCDATA)*>
<!ELEMENT IncludeFile (#PCDATA)*>
<!ELEMENT Uuid (#PCDATA)*>

<!--********************* Composite File ************************* -->
<!ELEMENT CompositeFile (Comments| IncludeFile| Module | BOArtifact |
Constant | Enumeration | Exception | Structure | Typedef | Union | Uuid)*>
<!ATTLIST CompositeFile Name CDATA #IMPLIED>

<!-- ************************** Module *************************** -->
<!ELEMENT Module (#PCDATA | BOArtifact | DOArtifact | Comments | Constant |
Enumeration | Exception | Structure | Typedef | Union | Uuid)*>
<!ATTLIST Module Name CDATA #IMPLIED>
```

```
<!-- ***************************** DO File *************************** -->
<!ELEMENT DOFile (Comments | IncludeFile | Module | DOArtifact |
Constant | Enumeration | Exception | Structure | Typedef | Union | Uuid)*>
<!ATTLIST DOFile
  Name CDATA #IMPLIED>

<!-- ************************** BOArtifact ********************** -->
<!ELEMENT BOArtifact (Interface| KeyArtifact | CopyArtifact | BOImplementation)*>
<!ELEMENT   Interface   (#PCDATA   |   IsQueryable   |   IsWLM   |   Comments   |
InterfaceInheritance | Attribute | Method | Constant | Enumeration | Exception
| Structure | Typedef | Union | Uuid |ObjectRelationship | CompositeInfo)*>
<!ATTLIST Interface
  Name CDATA #IMPLIED>
<!ELEMENT IsQueryable (#PCDATA)*>
<!ELEMENT IsWLM (#PCDATA)*>
<!ELEMENT InterfaceInheritance (#PCDATA)*>

<!-- ************************** CompositeInfo ********************* -->
<!ELEMENT CompositeInfo (CompositeRules | InterfaceReference)*>
<!ATTLIST CompositeInfo
          CompositeToolMoniker CDATA #IMPLIED>

<!--*************************** CompositeRules ********************** -->
<!ELEMENT CompositeRules (#PCDATA)*>

<!--*********************** InterfaceReference ******************** -->
<!ELEMENT InterfaceReference EMPTY>
```

```
<!ATTLIST InterfaceReference
          Referenced CDATA #IMPLIED
          InstanceInterface CDATA #IMPLIED
          InstanceName CDATA #IMPLIED
          InstanceKind CDATA #IMPLIED>

<!--*********************** DataReference ************************** -->
<!ELEMENT DataReference (Prereq | DataStore | Chain | Select)*>
<!ATTLIST DataReference
          Type CDATA #IMPLIED
          ReferencedCategory CDATA #IMPLIED
          ReferencedInterface CDATA #IMPLIED
          ReferencedName CDATA #IMPLIED
          ReferencedKind CDATA #IMPLIED
          InstanceInterface CDATA #IMPLIED
          InstanceName CDATA #IMPLIED
          InstanceKind CDATA #IMPLIED>

<!-- ***************************** Prereq *************************** -->
<!ELEMENT Prereq (Prereq | DataStore | Chain | Select)*>
<!ATTLIST Prereq
          ReferencedParameterName CDATA #IMPLIED
          ReferencedParameterKind CDATA #IMPLIED
          TargetInterface CDATA #IMPLIED
          OperationName CDATA #IMPLIED
          OperationKind CDATA #IMPLIED
          ParameterName CDATA #IMPLIED
          ParameterKind CDATA #IMPLIED
          Helper CDATA #IMPLIED
          ReferencedCategory CDATA #IMPLIED
          ReferencedInterface CDATA #IMPLIED
          ReferencedName CDATA #IMPLIED
          ReferencedKind CDATA #IMPLIED
          InstanceInterface CDATA #IMPLIED
          InstanceName CDATA #IMPLIED
          InstanceKind CDATA #IMPLIED>

<!--**************************** DataStore *************************** -->
<!ELEMENT DataStore (#PCDATA)*>
<!ATTLIST DataStore
          ReferencedCategory CDATA #IMPLIED
          ReferencedInterface CDATA #IMPLIED
          ReferencedName CDATA #IMPLIED
          ReferencedKind CDATA #IMPLIED
          InstanceInterface CDATA #IMPLIED
          InstanceName CDATA #IMPLIED
          InstanceKind CDATA #IMPLIED>

<!-- ****************************** Chain *************************** -->
<!ELEMENT Chain EMPTY>

<!-- ****************************** Select *************************** -->
```

```
<!ELEMENT Select (NullAttribute)*>

<!--************************* NullAttribute ********************** -->
<!ELEMENT NullAttribute (Prereq | DataStore | Chain | Select)*>
<!ATTLIST NullAttribute
          AttributeInterface CDATA #IMPLIED
          AttributeName CDATA #IMPLIED
          AttributeKind CDATA #IMPLIED
          Negated CDATA #IMPLIED
          ReferencedCategory CDATA #IMPLIED
          ReferencedInterface CDATA #IMPLIED
          ReferencedName CDATA #IMPLIED
          ReferencedKind CDATA #IMPLIED
          InstanceInterface CDATA #IMPLIED
          InstanceName CDATA #IMPLIED
          InstanceKind CDATA #IMPLIED>

<!--************************** DelegateToGroup ********************** -->
<!ELEMENT DelegateToGroup (#PCDATA)*>


<!--************************* KeyAttrMapGroup ********************* -->
<!ELEMENT KeyAttrMapGroup (KeyAttrMapInfo)*>

<!--************************* KeyAttrMapInfo ********************* -->
<!ELEMENT KeyAttrMapInfo (KeyAttrReference)*>
<!ATTLIST KeyAttrMapInfo
```

```
          Interface CDATA #IMPLIED
          AttributeName CDATA #IMPLIED
          AttributeKind CDATA #IMPLIED>

<!--************************ KeyAttrReference ********************* -->
<!ELEMENT KeyAttrReference (KeyAttributeReference |
 InstanceAttributeReference)*>

<!--*********************** KeyAttributeReference ******************* -->
<!ELEMENT KeyAttributeReference EMPTY>
<!ATTLIST KeyAttributeReference
          Interface CDATA #IMPLIED
          AttributeName CDATA #IMPLIED
          AttributeKind CDATA #IMPLIED>

<!--****************** InstanceAttributeReference ******************* -->
<!ELEMENT InstanceAttributeReference EMPTY>
<!ATTLIST InstanceAttributeReference
          Interface CDATA #IMPLIED
          AttributeName CDATA #IMPLIED
          AttributeKind CDATA #IMPLIED>

<!-- ********************** LocationGroup ************************** -->
<!ELEMENT LocationGroup (LocationByKey)*>

<!--*********************** LocationByKey ************************* -->
```

```
<!ELEMENT LocationByKey (InterfaceReference)*>
<!ATTLIST LocationByKey
          Group CDATA #IMPLIED
          FindObject CDATA #IMPLIED
          CreateObject CDATA #IMPLIED
          RemoveObject CDATA #IMPLIED
          CreateFromCopy CDATA #IMPLIED
          HomeName CDATA #IMPLIED
          FactoryFinderName CDATA #IMPLIED
          HomeParameter CDATA #IMPLIED>

<!--********************** Object Relationship ******************* -->
<!ELEMENT ObjectRelationship (Uuid | ObjectType |
ReferenceCollectionImplementation | ReferenceResolvedByForeignKey | HomeToQuery
|Attribute | Method)*>
<!ATTLIST ObjectRelationship Name CDATA #IMPLIED>
<!ELEMENT ObjectType (#PCDATA)*>
<!ELEMENT ReferenceCollectionImplementation (#PCDATA)*>
<!ELEMENT ReferenceResolvedByForeignKey (#PCDATA)*>

<!-- *************************** Construct ************************ -->
<!ELEMENT Constant (Type | Initializer | Size | Uuid)*>
<!ATTLIST Constant
  Name CDATA #IMPLIED>

<!ELEMENT Enumeration (Member | Uuid)*>
<!ATTLIST Enumeration
```

```
 Name CDATA #IMPLIED>

<!ELEMENT Exception (Member | Uuid)*>

<!ATTLIST Exception Name CDATA #IMPLIED>


<!ELEMENT Member (Type | Size | TypeSpecification | SequenceSize | ArraySize |
CaseLabel | Uuid)*>

<!ATTLIST Member Name CDATA #IMPLIED>

<!ELEMENT TypeSpecification (#PCDATA)*>

<!ELEMENT SequenceSize (#PCDATA)*>

<!ELEMENT ArraySize (#PCDATA)*>

<!ELEMENT CaseLabel (#PCDATA)*>

<!ELEMENT Structure (Member | Uuid)*>

<!ATTLIST Structure Name CDATA #IMPLIED>

<!ELEMENT Typedef (Type | Size | TypeSpecification | SequenceSize | ArraySize |
Uuid)*>

<!ATTLIST Typedef Name CDATA #IMPLIED>

<!ELEMENT Union (SwitchSpecifier | Member | Uuid)*>

<!ATTLIST Union Name CDATA #IMPLIED>

<!ELEMENT SwitchSpecifier (#PCDATA)*>


<!-- **************************** Attribute ************************** -->

<!ELEMENT Attribute (Type| Initializer| Size| Implementation| Read-only|
Overridable| (StringBehaviour?) | GetterMethodBodyForPlatform |
SetterMethodBodyForPlatform | GetterImplementationType | SetterImplementationType
| ParentInterface | DOAttributeName | Uuid | DataReference | DelegateToGroup|
UsedInImplementation | StaticData |

GetterBodiesKeepAllPlatformsInSyncCpp |

GetterBodiesKeepAllPlatformsInSyncJava |
SetterBodiesKeepAllPlatformsInSyncCpp |
SetterBodiesKeepAllPlatformsInSyncJava)*>
<!ATTLIST Attribute
  Name CDATA #IMPLIED>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT Initializer (#PCDATA)*>
<!ELEMENT Size (#PCDATA)*>
<!ELEMENT Implementation (#PCDATA)>
<!ELEMENT Read-only (#PCDATA)>
<!ELEMENT Overridable (#PCDATA)>
<!ELEMENT StringBehaviour (#PCDATA)>
<!ELEMENT GetterImplementationType (#PCDATA)*>
<!ELEMENT SetterImplementationType (#PCDATA)*>
<!ELEMENT DOAttributeName (#PCDATA)*>
<!ELEMENT ParentInterface (#PCDATA)*>
<!ELEMENT UsedInImplementation (#PCDATA)*>
<!ELEMENT StaticData (#PCDATA)*>
<!ELEMENT GetterBodiesKeepAllPlatformsInSyncCpp (#PCDATA)*>
<!ELEMENT GetterBodiesKeepAllPlatformsInSyncJava (#PCDATA)*>
<!ELEMENT SetterBodiesKeepAllPlatformsInSyncCpp (#PCDATA)*>
<!ELEMENT SetterBodiesKeepAllPlatformsInSyncJava (#PCDATA)*>
<!ELEMENT GetterMethodBodyForPlatform (Language)*>
<!ATTLIST GetterMethodBodyForPlatform Name CDATA #IMPLIED>
<!ELEMENT SetterMethodBodyForPlatform (Language)*>
<!ATTLIST SetterMethodBodyForPlatform Name CDATA #IMPLIED>
<!--ELEMENT ImplementationLanguage ANY-->
```

```
<!ELEMENT Language (UseToolDef, UseFile, UseTDE,Filename?, ToolDef?,
UserDef?,((OOSQLSourceType?, OOSQLUserDef?) | (ESQLSourceType?, ESQLUserDef?))?)>

<!ATTLIST Language

  Name CDATA #IMPLIED>

<!ELEMENT UseToolDef (#PCDATA)>

<!ELEMENT UseFile (#PCDATA)>

<!ELEMENT UseTDE (#PCDATA)>

<!ELEMENT Filename (#PCDATA)*>

<!ELEMENT ToolDef (#PCDATA)*>

<!ELEMENT UserDef (#PCDATA)*>

<!ELEMENT OOSQLSourceType (#PCDATA)*>

<!ELEMENT OOSQLUserDef (#PCDATA)*>

<!ELEMENT ESQLSourceType (#PCDATA)*>

<!ELEMENT ESQLUserDef (#PCDATA)*>


<!--**************************** Method ************************* -->

<!-- Preserve spaces and newline -->

<!ATTLIST Filename xml:space (default|preserve) "preserve">

<!ATTLIST ToolDef xml:space (default|preserve) "preserve">

<!ATTLIST UserDef xml:space (default|preserve) "preserve">

<!ATTLIST Comments xml:space (default|preserve) "preserve">

<!ATTLIST OOSQLUserDef xml:space (default|preserve) "preserve">

<!ATTLIST ESQLUserDef xml:space (default|preserve) "preserve">

<!-- Preserve spaces and newline -->

<!ELEMENT Method (ReturnType | Size | Implementation| MethodBodyForPlatform |
One-way | Overridable | ConstMethod | IsFrameworkMethod | ExceptionRaises |
Context | Parameter | Uuid | ParentInterface | DataReference | DelegateToGroup
| MethodType | UsedInImplementation | StaticMethod | PushDownMethod |
MappedMethod | PlatformSet | MethodBodiesKeepAllPlatformsInSyncCpp |
MethodBodiesKeepAllPlatformsInSyncJava)*>
<!ATTLIST Method
  Name CDATA #IMPLIED>
<!ELEMENT ReturnType (#PCDATA)>
<!ELEMENT ConstMethod (#PCDATA)>
<!ELEMENT PushDownMethod (#PCDATA)*>
<!ELEMENT MappedMethod (#PCDATA)*>
<!ELEMENT MethodBodyForPlatform (Language)*>
<!ATTLIST MethodBodyForPlatform Name CDATA #IMPLIED>
<!ELEMENT ImplementationType (#PCDATA)*>
<!ELEMENT Designpattern (#PCDATA)*>
<!ELEMENT UseMacro (#PCDATA)>
<!ELEMENT One-way (#PCDATA)>
<!ELEMENT IsFrameworkMethod (#PCDATA)>
<!ELEMENT ExceptionRaises (#PCDATA)*>
<!ELEMENT Context (#PCDATA)*>
<!ELEMENT StaticMethod (#PCDATA)*>
<!ELEMENT PlatformSet (#PCDATA)>
<!ELEMENT MethodBodiesKeepAllPlatformsInSyncCpp (#PCDATA)*>
<!ELEMENT MethodBodiesKeepAllPlatformsInSyncJava (#PCDATA)*>
<!ELEMENT Parameter (ParameterType | Size | DirectionalAttr | Uuid )*>
<!ATTLIST Parameter
  Name CDATA #IMPLIED>
<!ELEMENT ParameterType (#PCDATA)*>
<!ELEMENT DirectionalAttr (#PCDATA)*>
```

```
<!ELEMENT MethodType (#PCDATA)*>


<!--*********************** DOArtifact ***************************** -->

<!ELEMENT DOArtifact (DOInterface| Comments)*>


<!--************************** Key ********************************** -->

<!ELEMENT KeyArtifact (#PCDATA| File | Module | Interface | Attribute | Method
| InterfaceInheritance | Uuid | Prologue | CppPrologue | JavaPrologue | Epilogue
| CppEpilogue | JavaEpilogue | KeyAttrMapGroup)*>

<!ELEMENT File (#PCDATA | Uuid | IncludeFile | Comments )*>

<!ATTLIST File Name CDATA #IMPLIED>


<!--************************** Copy ********************************* -->

<!ELEMENT CopyArtifact (#PCDATA | File | Module | Interface | Attribute | Method
| InterfaceInheritance | Uuid | Prologue | CppPrologue | JavaPrologue| Epilogue
| CppEpilogue | JavaEpilogue)*>


<!--************************** BO Implementation ******************* -->

<!ELEMENT BOImplementation (#PCDATA| File | Module | Interface | Pattern |
LazyEvaluation | DataObjectInterface | SessionableBO | ImplementationLanguage |
UserData | ImplementationInheritance | Attribute | Method | Prologue |
CppPrologue | JavaPrologue | Epilogue | CppEpilogue | JavaEpilogue | Key | Copy
| Handle | DOInterface | MO | Uuid | LocationGroup | ObjectRelationship)*>

<!ELEMENT Pattern (#PCDATA)>

<!ELEMENT LazyEvaluation (#PCDATA)>

<!ELEMENT DataObjectInterface (#PCDATA)>

<!ELEMENT SessionableBO (#PCDATA)>

<!ELEMENT ImplementationLanguage (#PCDATA)*>

<!ELEMENT UserData (#PCDATA)*>

<!ELEMENT ImplementationInheritance (#PCDATA)>
```

```
<!ELEMENT Prologue (#PCDATA)*>
<!ELEMENT CppPrologue (#PCDATA)*>
<!ELEMENT JavaPrologue (#PCDATA)*>
<!ELEMENT Epilogue (#PCDATA)*>
<!ELEMENT CppEpilogue (#PCDATA)*>
<!ELEMENT JavaEpilogue (#PCDATA)*>
<!ELEMENT Key (#PCDATA)*>
<!ELEMENT Handle (#PCDATA)*>

<!-- Preserve spaces and newline -->
<!ATTLIST Prologue xml:space (default|preserve) "preserve">
<!ATTLIST CppPrologue xml:space (default|preserve) "preserve">
<!ATTLIST JavaPrologue xml:space (default|preserve) "preserve">
<!ATTLIST Epilogue xml:space (default|preserve) "preserve">
<!ATTLIST CppEpilogue xml:space (default|preserve) "preserve">
<!ATTLIST JavaEpilogue xml:space (default|preserve) "preserve">

<!--************************* DO Stuff ***************************** -->
<!ELEMENT  DOInterface  (File  |  Module  |  Interface  |  Comments  |
InterfaceInheritance | DOImplementation | Attribute | Method | Constant |
Enumeration | Exception | Structure | Typedef | Union | Uuid)*>
<!ATTLIST DOInterface
  Name CDATA #IMPLIED>

<!--*************************** DO Implementation ****************** -->
<!ELEMENT DOImplementation (#PCDATA | File | Module | Interface | Environment |
PersistentBehaviour | DataAccessPattern | PointerHandle | Key | Copy |
ImplementationInheritance | Attribute | Method | Prologue | CppPrologue |
JavaPrologue | Epilogue | CppEpilogue | JavaEpilogue | PO | POMapping | Uuid)*>

<!ELEMENT Environment (#PCDATA)*>

<!ELEMENT PersistentBehaviour (#PCDATA)*>

<!ELEMENT DataAccessPattern (#PCDATA)*>

<!ELEMENT PointerHandle (#PCDATA)*>

<!ELEMENT Copy (#PCDATA)*>


<!--************************** PO Stuff ************************** -->

<!ELEMENT PO (Uuid?, ShortName?, ModuleName?, JavaPackage?, JavaPath?,

PAOService?, POAccessType, Prologue?, CppPrologue?, JavaPrologue?, Epilogue?,
CppEpilogue?, JavaEpilogue?, BindFileName, Comments?, POAttribute*, POMethod*,
Schema?)>

<!ATTLIST PO

  Name CDATA #IMPLIED>

<!ELEMENT POAccessType (#PCDATA)>

<!ELEMENT BindFileName (#PCDATA)*>

<!ELEMENT POAttribute (Uuid?, Type, Size, ColumnName, KeySequenceNumber?, Key?,
Getter?, Setter?, GetterBodiesKeepAllPlatformsInSyncCpp?,
SetterBodiesKeepAllPlatformsInSyncCpp?, GetterMethodBodyForPlatform*,
SetterMethodBodyForPlatform*)>

<!ATTLIST POAttribute Name CDATA #IMPLIED>

<!ELEMENT KeySequenceNum (#PCDATA)*>

<!ELEMENT KeySequenceNumber (#PCDATA)*>

<!ELEMENT POMethod (Uuid?, IsFrameworkMethod, MethodFunction?, Implementation,
ReturnType, POParameter*, PlatformSet?, MethodBodiesKeepAllPlatformsInSyncCpp?,
MethodBodyForPlatform*)>

<!ATTLIST POMethod Name CDATA #IMPLIED>

<!ELEMENT POParameter (Type, Size?, ParmAccessType?)>

<!ATTLIST POParameter Name CDATA #IMPLIED>
```

```
<!ELEMENT ParmAccessType (#PCDATA)>


<!--************************ DOImpl->PO Mappings ******************** -->

<!ELEMENT POMapping (AttributeMapping*, MethodMapping*)>

<!ELEMENT AttributeMapping (DataObjectAttribute, PatternType?,
PrimitiveMapping?,((POAttributeImplementation+, MappingHelper?) | (KeyMapping |
StructMapping)*)?)>

<!ELEMENT DataObjectAttribute (#PCDATA)>

<!ELEMENT PatternType (#PCDATA)>

<!ELEMENT PrimitiveMapping (POAttributeImplementation+, MappingHelper?)>

<!ELEMENT POAttributeImplementation (POInstance, POInstanceAttribute)>

<!ELEMENT POInstance (#PCDATA)>

<!ELEMENT POInstanceAttribute (#PCDATA)>

<!ELEMENT MappingHelper (MappingHelperClass, MappingHelperToTarget,
MappingHelperToSource)*>

<!ELEMENT MappingHelperClass (#PCDATA)*>

<!ELEMENT MappingHelperToTarget (#PCDATA)*>

<!ELEMENT MappingHelperToSource (#PCDATA)*>

<!ELEMENT KeyMapping (KeyFullName, HomeToQuery, (KeyAttributeMapping)*)>

<!ELEMENT KeyFullName (#PCDATA)>

<!ELEMENT HomeToQuery (#PCDATA)*>

<!ELEMENT KeyAttributeMapping (KeyAttribute, (POAttributeImplementation)*,
MappingHelper?)>

<!ELEMENT KeyAttribute (#PCDATA)>

<!ELEMENT StructMapping (StructFullName, (StructAttributeMapping)*)>

<!ELEMENT StructFullName (#PCDATA)>

<!ELEMENT StructAttributeMapping (StructAttribute, PrimitiveMapping?, (KeyMapping
| StructMapping)*)>
```

```
<!ELEMENT StructAttribute (#PCDATA)>

<!ELEMENT MethodMapping (DataObjectMethod, AlwaysCompletes,

                        (POMethodImplementation)*)>

<!ELEMENT POMethodImplementation (POInstance, POInstanceMethod)>

<!ELEMENT DataObjectMethod (#PCDATA)*>

<!ELEMENT AlwaysCompletes (#PCDATA)*>

<!ELEMENT POInstanceMethod (#PCDATA)>


<!--************************** Schema Stuff ************************ -->

<!ELEMENT Schema (((DatabaseName, UserName?, SchemaName, ContainedBySchemaGroup,
DatabaseType?, SchemaType, PKConstraint?, PKConstraintComment?, Comments?,
SQLFilename, Column*,((ForeignKey*) |(ViewSelect?, ViewFrom?, ViewWhere?,
ViewOrderby?, ViewGroupby?, ViewHaving?, IsUserWrittenClause?, ViewSchema+,
ViewColumnMapping?))) |(SchemaName, ShortName, ModuleName, SchemaType,
JavaPackage, JavaPath, JavaClassName, PAOService, Property*, SchemaMethod*)),
PO*)>

<!ATTLIST Schema

  Name CDATA #IMPLIED>

<!ELEMENT ContainedBySchemaGroup (#PCDATA)*>

<!ELEMENT DatabaseName (#PCDATA)>

<!ELEMENT UserName (#PCDATA)*>

<!ELEMENT SchemaName (#PCDATA)>

<!ELEMENT ShortName (#PCDATA)>

<!ELEMENT ModuleName (#PCDATA)>

<!ELEMENT SchemaType (#PCDATA)>

<!ELEMENT PKConstraint (#PCDATA)*>

<!ELEMENT PKConstraintComment (#PCDATA)*>

<!ELEMENT SQLFilename (#PCDATA)*>

<!ELEMENT JavaPackage (#PCDATA)>
```

```
<!ELEMENT JavaPath (#PCDATA)>

<!ELEMENT JavaClassName (#PCDATA)>

<!ELEMENT PAOService (#PCDATA)>

<!ELEMENT SchemaMethod (Uuid?, IsFrameworkMethod, MethodFunction?,
Implementation, ReturnType, SchemaParameter*)>

<!ATTLIST SchemaMethod Name CDATA #IMPLIED>

<!ELEMENT MethodFunction (#PCDATA)*>

<!ELEMENT SchemaParameter (Type, Size?)>

<!ATTLIST SchemaParameter Name CDATA #IMPLIED>

<!ELEMENT Column (ColumnName, ColumnType, NotNull, KeySequenceNumber,
ColumnSequenceNumber, Length, Scale, ForBitData, Comments)>

<!ATTLIST Column

  Name CDATA #IMPLIED>

<!ELEMENT ColumnName (#PCDATA)>

<!ELEMENT ColumnType (#PCDATA)>

<!ELEMENT NotNull (#PCDATA)>

<!ELEMENT ColumnSequenceNumber (#PCDATA)*>

<!ELEMENT Length (#PCDATA)>

<!ELEMENT Scale (#PCDATA)>

<!ELEMENT ForBitData (#PCDATA)>

<!ELEMENT Property (TypeString, TypeQualifier, Size, Key?, Getter?, Setter?,
Uuid?)>

<!ATTLIST Property

  Name CDATA #IMPLIED>

<!ELEMENT TypeString (#PCDATA)>

<!ELEMENT TypeQualifier (#PCDATA)>

<!ELEMENT Getter (#PCDATA)>
```

```
<!ELEMENT Setter (#PCDATA)>
<!ELEMENT ForeignKey (IsUnnamed, ForeignKeyName?, TargetSchema,
                      Comments, FKMapping*)>
<!ATTLIST ForeignKey Name CDATA #IMPLIED>
<!ELEMENT ForeignKeyName (#PCDATA)*>
<!ELEMENT IsUnnamed (#PCDATA)>
<!ELEMENT TargetSchema (DatabaseName, UserName?, SchemaName,
ContainedBySchemaGroup)>
<!ELEMENT FKMapping (OwningAttribute, TargetAttribute)>
<!ELEMENT OwningAttribute (#PCDATA)>
<!ELEMENT TargetAttribute (#PCDATA)>
<!ELEMENT ViewSelect (#PCDATA)*>
<!ELEMENT ViewFrom (#PCDATA)*>
<!ELEMENT ViewWhere (#PCDATA)*>
<!ELEMENT ViewOrderby (#PCDATA)*>
<!ELEMENT ViewGroupby (#PCDATA)*>
<!ELEMENT ViewHaving (#PCDATA)*>
<!ELEMENT IsUserWrittenClause (#PCDATA)*>
<!ELEMENT ViewSchema (DatabaseName, UserName?, SchemaName,
ContainedBySchemaGroup)>
<!ELEMENT ViewColumnMapping (ViewColumn*)>
<!ELEMENT ViewColumn (TargetSchema, TargetColumn)>
<!ATTLIST ViewColumn Name CDATA #IMPLIED>
<!ELEMENT TargetColumn (#PCDATA)*>


<!--************************ SchemaGroup Stuff ******************** -->
```

```
<!ELEMENT    SchemaGroup    (DatabaseName?,    DatabaseType?,    DDLFilename?,
EditGeneratedFile?, Schema*)>

<!ATTLIST SchemaGroup

  Name CDATA #IMPLIED>

<!ELEMENT DatabaseType (#PCDATA)*>

<!ELEMENT DDLFilename (#PCDATA)*>

<!ELEMENT EditGeneratedFile (#PCDATA)*>


<!--****************************** MO Stuff ************************** -->

<!ELEMENT MO (File | Module | Interface | ImplementationInheritance |

              MOApplicationAdaptor | Uuid)*>

<!ELEMENT MOApplicationAdaptor (#PCDATA)*>

<!-- UDBO ? -->
```

As shown above the meta data model in the form of the Document Type Definition (DTD) specifies the set of required and optional elements, and their attributes, for the XML documents which contain the data content of the objects. In addition, the DTD specifies the names of the tags and the relationships among elements in the XML document. It will also be appreciated that the task tree elements for the "User-Defined Business Objects" 120 are expressed using containment relationships in the Document Type Definition shown above. For example, the file element 121 is defined as <!ELEMENT BOFile(Comments|....|Uuid)*>, and the module element 122 is defined as <!ELEMENT Module (#PCDATA|....Uuid)*>. Similarly, in the DTD the "attributes" of the object are defined as <!ELEMENT Attribute (Type |Initializer|....)*>, and the "methods" of the object are defined as <!ELEMENT Method (ReturnType|.....)*>. The specific syntax in the meta data model as specified in the above DTD will be within the understanding of those familiar with XML.

According to the present invention, the data model comprises a task oriented structure which preserves the sequence of steps followed by a user during the creation of the object. Thus, the data model for the object, i.e. instance of data, is stored according to the meta data model in the exact order created by the user, and the data model thereby provides a data structure from which the user interface of the object builder tool can be inferred. Since the user interface is implicit in the organization of the data model, the data model can be imported by another tool by simply following or scripting the data structure.

In order to take advantage of the task oriented structure expressed in XML, a data import/export utility in accordance with the present invention is implemented to include an XML parser. In known manner, the XML parser is a validating parser which uses the Document Type Definition to validate each element in the XML document, i.e. the data model. In the present context, the validating XML parser parses the XML data document and builds a "document tree" which is then returned to the data import/export utility.

To export a data file, the data model is expressed as a structured XML document file which as described above mirrors the exact order in which the object was created by the user. Because the tasks are expressed in terms of containment relationships in the XML document as defined by the DTD (i.e. the meta data model), the importation, i.e. reading, of the XML document serves to extract the data model in the same sequence the data model was created. It will be appreciated that this approach eliminates the need for tags and other indicators of location as is the case for conventional data import/export files. The validation of the data model occurs implicitly as the XML document is read by the data import/export utility.

Reference is made to Fig. 2 which shows in flow chart form the principle steps performed by a data import/export utility according to the present invention. In this example, a task oriented data model according to the present invention is to be imported by the object builder tool 10 (described above with reference to Fig. 1). The first step as shown in Fig. 2 involves making a call to the data import utility (step 201). Next, the data import utility retrieves the XML document from memory (step 202) and passes the document to the validating XML parser (step 203). The XML parser then parses the XML document and builds a document tree for the XML document (step 204). Preferably, the XML parser is a validating parser, which means that each element appearing in the XML document is validated against the meta data model specified in the Document Type Definition. Upon completion of the parsing and validation of the XML document, the document tree is returned to the data import utility (step 205). The next step performed by the data import utility involves adding or updating the object according to the data model for each element contained in the document tree (step 206). After the object has been added or updated based on the contents of the XML document file, control returns to the object builder (i.e. application program).

The invention may be implemented as a computer program product comprising a program storage device and means recorded on said program storage device for instructing a computer to perform the steps of the invention and as a data structure embodied in a program storage device. Such a program storage device may include diskettes, optical discs, tapes, CD-ROMS, hard drives, memory including ROM or RAM, computer tapes or other storage media capable of storing a computer program.

The invention may also be implemented in a computer system. In a preferred embodiment, a system is provided comprising a computer program operating on a data processing system, with the computer program embodying the method of the invention and producing an output of the method on a display or output device. Data processing systems include computers, computer networks, embedded systems and other systems capable of executing a computer program. A computer includes a processor and a memory device and optionally, a storage device, a video display and/or an input device. Computers may equally be in stand-alone form (such as the traditional desktop personal computer) or integrated into another apparatus (such as a cellular telephone).

While this invention has been described in relation to preferred embodiments, it will be understood by those skilled in the art that changes in the details of processes and structures may be made without departing from the spirit and scope of this invention. Many modifications and variations are possible in light of the above teaching. Thus, it should be understood that the above described embodiments have been provided by way of example rather than as a limitation and that the specification and drawing are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A task oriented data structure embodied in a program storage device for an object oriented application program, the object oriented application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating one or more objects, said data structure comprising:

(a) a data model suitable for storage on a storage media;

(b) said data model including a plurality of data elements;

(c) each of said data elements corresponding to one of the tasks in said sequence of tasks; and

(d) said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface.

2. The task oriented data structure embodied in a program storage device as claimed in claim 1, wherein said meta data model includes means for validating each of said data elements and the arrangement of said data elements.

3. The task oriented data structure embodied in a program storage device as claimed in claim 1 or claim 2, wherein said data model is expressed in an Extensible Markup Language or XML, and said data elements being arranged according to containment constructs specified in said meta data model.

4. The task oriented data structure embodied in a program storage device as claimed in claim 3, wherein said means for validating comprises a Document Type Definition specified in XML.

5. In an application program for creating objects, the application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating the objects, and said application program including an import utility for importing document files, said import utility comprising:

(a) means for inputting a document file expressed in a meta data programming language, and wherein said document file comprises a plurality of data elements, each of said data elements corresponding to one of the tasks in said sequence of tasks, and said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface;

(b) parser means for parsing said document file according to said meta data programming language, and said parser means including means for creating a document tree, said document tree comprising a plurality of nodes, wherein each of said nodes corresponds to an item in the user interface; and

(c) means for updating the object according to the nodes contained in said document tree.

6. The import utility as claimed in claim 5, wherein said meta data model includes means for validating each of the nodes in said document tree.

7. The import utility as claimed in claim 5 or claim 6, wherein said document file comprises a text file expressed in Extensible Markup Language or XML, and said data elements being arranged according to containment constructs specified in said meta data model.

8. The import utility as claimed in claim 7, wherein said means for validating comprises a Document Type Definition specified in XML.

9. The import utility as claimed in any one of claims 5 to 8, wherein said document file includes scripting means for translating each of said data elements.

10. A computer program product for an application program for creating objects, the application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating the objects, and said application program including an import utility for importing document files, said computer program product comprising:

a program storage device;

means recorded on said program storage device for instructing a computer to perform the steps of,

(a) inputting a document file, wherein said document file is expressed according to a meta data programming language, said document file comprising a plurality of data elements, each of said data elements corresponding to one of the tasks in said sequence of tasks, and said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface;

(b) parsing said document file according to said meta data programming language;

(c) creating a document tree from said parsed document file, said document tree comprising a plurality of nodes, wherein each of said nodes corresponds to an item in the user interface; and

(d) updating the object according to the nodes contained in said document tree.

11. The computer program product as claimed in claim 10, further including validating means for use by said parsing means for validating each of the nodes in said document tree.

12. The computer program product claimed in claim 10 or claim 11, wherein said meta data language comprises Extensible Markup Language or XML, and said data elements being arranged according to containment constructs specified in said meta data model.

13. The computer program product as claimed in claim 12, wherein said means for validating comprises a Document Type Definition specified in XML.

14. A computer system for creating objects in an application program, the application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating the objects, and said application program including an import utility for importing document files, said computer system comprising:

(a) means for inputting a document file, wherein said document file is expressed according to a meta data programming language, said document file comprising a plurality of data elements, each of said data elements corresponding to one of the tasks in said sequence of tasks, and said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface;

(b) a parser for parsing said document file according to said meta data programming language;

(c) means for creating a document tree from said parsed document file, said document tree comprising a plurality of nodes, wherein each of said nodes corresponds to an item in the user interface; and

(d) means for updating the object according to the nodes contained in said document tree.

15. The computer system as claimed in claim 14, further including validating means for use by said parser for validating each of the nodes in said document tree.

16. The computer program product claimed in claim 14 or claim 15, wherein said meta data language comprises Extensible Markup Language or XML, and said data elements being arranged according to containment constructs specified in said meta data model.

17. The computer program product as claimed in claim 16, wherein said validating means comprises a Document Type Definition specified in XML.

18. The computer program product as claimed in any one of claims 14 to 17, wherein said data model includes scripting means for translating each of the data elements in said document file.

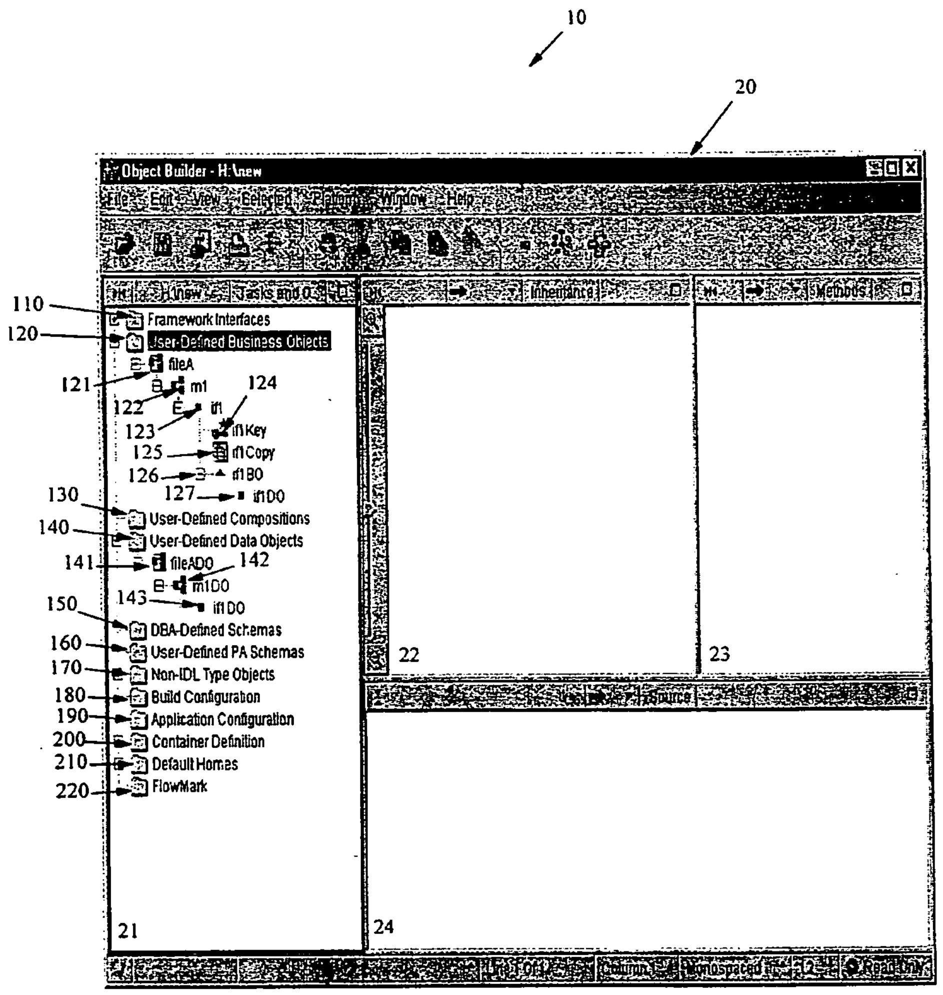

10
20
Object Builder - H:\new
File Edit View Selected Platform Window Help
H:\new Tasks and O
Inheritance
Methods
Source
110
Framework Interfaces
120
User-Defined Business Objects
fileA
121
m1
122
124
if1
123
if1Key
125
if1Copy
126
if1BO
127
if1DO
130
User-Defined Compositions
140
User-Defined Data Objects
141
fileADO
142
m1DO
143
if1DO
150
DBA-Defined Schemas
160
User-Defined PA Schemas
170
Non-IDL Type Objects
180
Build Configuration
190
Application Configuration
200
Container Definition
210
Default Homes
220
FlowMark
21
22
23
24
Column
Monospaced
Read Only

FIGURE 1

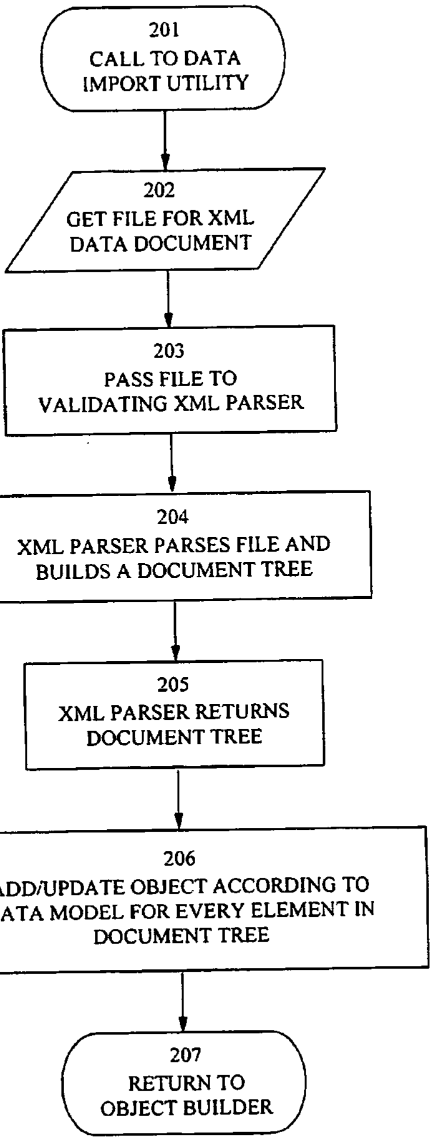

201
CALL TO DATA IMPORT UTILITY
202
GET FILE FOR XML DATA DOCUMENT
203
PASS FILE TO VALIDATING XML PARSER
204
XML PARSER PARSES FILE AND BUILDS A DOCUMENT TREE
205
XML PARSER RETURNS DOCUMENT TREE
206
ADD/UPDATE OBJECT ACCORDING TO DATA MODEL FOR EVERY ELEMENT IN DOCUMENT TREE
207
RETURN TO OBJECT BUILDER

FIGURE 2